2.1 Introduction to Object-Oriented Programming

- Defining a class allows us to :
  - Specify the structure of data precisely
  - Control operations performed on data so its always well-formed

DEFINING A CLASS: ATTRIBUTES

CLASS:

- ***Blueprint / Template** *
- Block of code that defines a type of data
- Have arbitrary # of attributes / methods
- Attributes can be of different types
- Name of class : Capitalized

INSTANCE OF A CLASS:

- An object that is part of a class
- Can hold collection of data bundled together (**instance attributes**)
- *Everything in Python is an object !*

INSTANCE ATRIBUTES:

- Individual piece of data in an instance

DOCSTRING:

1. Gives a description of the Class & its instance attributes
2. Syntax for declaring instance attributes: `<attribute_name>: <attribute_type>`
   - These annotations don't affect the code, but automated tools use them to help us write code & find bugs
   - Annotations in Docstring (meaning) and below it (types)!

```
from datetime import date # We are using a library to represent dates


class Tweet:
    """A tweet, like in Twitter.

    === Attributes ===
    content: the contents of the tweet.
    userid: the id of the user who wrote the tweet.
    created_at: the date the tweet was written.
    likes: the number of likes this tweet has received.
    """
    # Attribute types
    userid: str
    created_at: date
    content: str
    likes: int
```

1.

2.

CREATING AN INSTANCE OF A CLASS:

- Defining a new type:
  1. Importing the class
  2. Creating a new instance: creates a new object and stores its reference in the variable
     - `>>> tweet = Tweet()`

DEFINING AN INITIALIZER:

INITIALIZER:

- TO CREATE AND INITIALIZE THE INSTANCE ATTRIBUTES
- **An initializer never has a return statement, or always returns None.**
- **self:**
  - First parameter of initializer
  - Refers to instance that will be initialized & been created
  - Never annotate it (*its the class it belongs to*)
  - Never pass a value for self
  - Automatically receives id of instance to be initialized

```
class Tweet:
    # previous content omitted for brevity

    def __init__(self, who: str, when: date, what: str) -> None:
        """Initialize a new Tweet.
        """
```

- _ _ init _ _ is called automatically
- Values in parentheses are passed to it

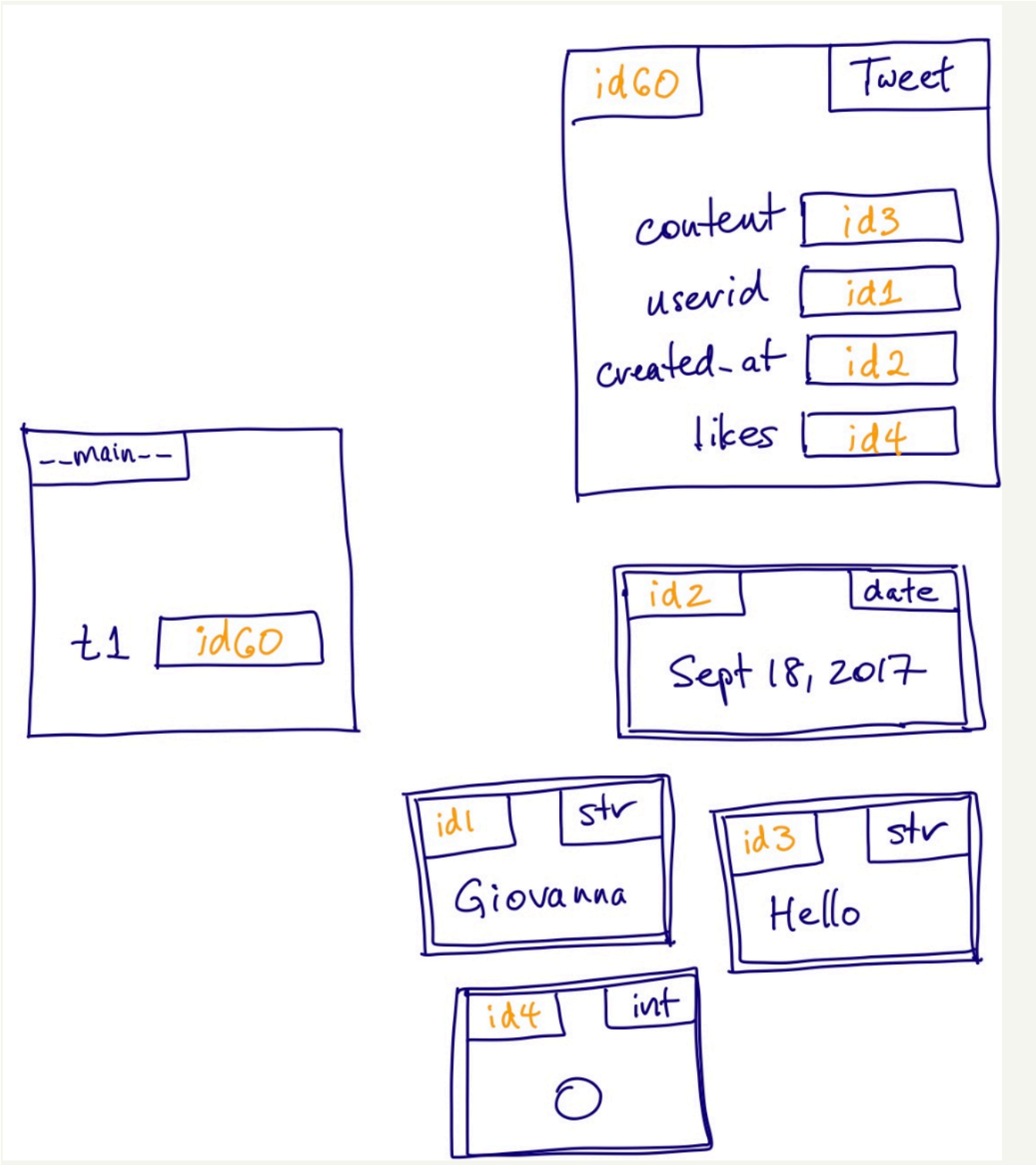
```
>>> from datetime import date
>>> t1 = Tweet('Giovanna', date(2018, 9, 18), 'Hello')
```

CREATING INSTANCE ATTRIBUTES:

- The “default" values

```
class Tweet:
    # previous content omitted for brevity...

    def __init__(self, who: str, when: date, what: str) -> None:
        """Initialize a new Tweet.
        """
        self.userid = who
        self.content = what
        self.created_at = when
        self.likes = 0
```



- With the new object set up and a reference to it stored, we can access each of its attributes using dot notation
- ```
  >>> from datetime import date
  >>> t1 = Tweet('Giovanna', date(2017, 9, 18), 'Hello
  ```
  ```

```
')
>>> t1.userid
'Giovanna'
>>> t1.created_at
datetime.date(2017, 9, 18)
>>> t1.content
'Hello'
>>> t1.likes
0
```

WHEN WE CREATE A NEW OBJECT:

1. Create new object (instance of the class)
2. Call _ _ init_ _ with new object passed to the parameter *self* , along with the other arguments
3. Return the new object

DEFINING A CLASS: METHODS:

- Functions defined within/ associated with a class
- First parameter : self (refers to object we are operating on)
- Separate entities from class itself
- To use them outside of the class you need to import them

```
class Tweet:
    ...

    def like(self, n: int) -> None:
        """Record the fact that <self> received <n> lik
es.

        Precondition: n >= 0
        """
        self.likes += n
```

- To use them outside of the class you need to import them
- We call the methods using dot notation

```
>>> from datetime import date
>>> tweet = Tweet('Rukhsana', date(2017, 9, 16), 'Hey!'
)
>>> tweet.like(10)  # dot notation!
>>> tweet.likes
10
```

METHODS vs FUNCTIONS:

METHODS:

- Part of class' definition
- Form basis of how others can use the class
- For most users of the class

FUNCTIONS:

- That operate on a class instance need to be imported separately before they are used
- Users of the class must implement themselves for their specific needs

SPECIAL METHODS:

- Called behind the scenes, you don't call them using regular method call syntax
- Defined with double underscores around them
- _ _ init _ _
- https://www.python-course.eu/python3_magic_methods.php